1. Какие классы содержаться с System.IO?

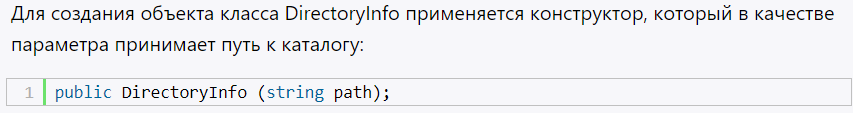
DriveInfo, Directory, DirectoryInfo, File, FileInfo, StreamReader, StreamWriter, BinaryWriter, BinaryReader, Path, MemoryStream, FileStream, Stream.

1. Для чего используются классы Directory и DirectoryInfo? В чем отличие?

DirectoryInfo - Данный класс предоставляет функциональность для создания, удаления, перемещения и других операций с каталогами. Во многом он похож на Directory, но не является статическим.

Directory - статический класс Directory предоставляет ряд методов для управления каталогами.

Если надо совершить одну-две операции с одним каталогом, то проще использовать класс Directory. Если необходимо выполнить последовательность операций с одним и тем же каталогом, то лучше воспользоваться классом DirectoryInfo. Почему? Дело в том, что методы класса Directory выполняют дополнительные проверки безопасности. А для класса DirectoryInfo такие проверки не всегда обязательны.



1. Для чего используются классы File и FileInfo? Какие методы они содержат.

Для работы с файлами. С их помощью мы можем создавать, удалять, перемещать файлы, получать их свойства и многое другое. вып. операции при пом. стат. методов (File), при пом. экземплярных методов (FileInfo)

**File:**

**Copy()** – копирует файл в новое место  
**Create()** – созд. файл  
**Delete()** – удал. файл  
**Move** – перемещ. файл в новое место   
**Exists(file)** – опр., сущ. ли файл

**FileInfo:**

**CopyTo(path)**: копирует файл в новое место по указанному пути path

**Create()**: создает файл

**Delete()**: удаляет файл

**MoveTo(destFileName)**: перемещает файл в новое место

Свойство **Directory**: получает родительский каталог в виде объекта DirectoryInfo

Свойство **DirectoryName**: получает полный путь к родительскому каталогу

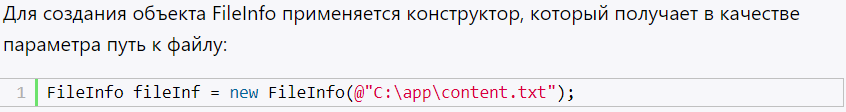
Свойство **Exists**: указывает, существует ли файл

Свойство **Length**: получает размер файла

Свойство **Extension**: получает расширение файла

Свойство **Name**: получает имя файла

Свойство **FullName**: получает полное имя файла



4. Для чего используются классы StreamReader и StreamWriter?

Для ввода и вывода информации в строковом представлении.

**StreamReader:**

**Close** – закр. считываемый файл и освобожд. все ресурсы  
**Peek** – возвр. след. доступный символ или -1  
**Read** – счит. и возвр. след. символ в численном предст. Read(char[] array, int index, int count)  
**ReadLine** – счит. одну строку в файле   
**ReadToEnd** – счит. весь текст из файла

**StreamWriter:**

**Close** – закр. записываемый файл и освобожд. все ресурсы  
**Flush** – запис. в файл оставшиеся в буфере д-е и очищ. буфер  
**Write** – запис. в файл д-е простейших типов, как int, double, char, string…  
**WriteLine** – также запис. д-е, добав. в файл символ окончания строки

5. Для чего используются классы BinaryWriter и BinaryReader?

Для ввода и вывода примитивных типов в двоичном формате.

**BinaryWriter:**

**Close()** – закр. поток и освобожд. ресурсы  
**Flush()**  – очищ. буфер, дописывая из него ост. д-е в файл  
**Seek()** – устан. позицию в потоке  
**Write()** – запис. д-е в поток

**BinaryReader:**

**Close()** – закр. поток и освобожд. ресурсы  
**ReadBoolean()** – счит. значение Boolean и перемещ. указ. на 1 байт  
**ReadByte()** – счит. 1 байт, перем. указ. на 1 байт  
**ReadChar()** – байт ск. в тек. кодировке  
**ReadDecimal()** – 16  
**ReadDouble()** – 8  
**ReadInt16()** – 2  
**ReadInt32()** – 4  
**ReadInt64()** – 8  
**ReadSingle()** – 4  
**ReadString()** –

6. Как можно сжимать и восстанавливать файлы?

Сжать класс DeflateStream и восстановить класс GZipStream

Статический класс **ZipFile** из простанства имен System.IO.Compression предоставляет дополнительные возможности для создания архивов. Он позволяет создавать архив из каталогов. Его основные методы:

* **void CreateFromDirectory(string sourceDirectoryName, string destinationFileName)**: архивирует папку по пути sourceDirectoryName в файл с названием destinationFileName
* **void ExtractToDirectory(string sourceFileName, string destinationDirectoryName)**: извлекает все файлы из zip-файла sourceFileName в каталог destinationDirectoryName

7. Расскажите алгоритм сжатия GZip.

Для создания объекта GZipStream можно использовать один из его конструкторов:

* GZipStream(Stream stream, CompressionLevel level): stream представляет данные, а level задает уровень сжатия
* GZipStream(Stream stream, CompressionMode mode): mode указывает, будут ли данные сжиматься или, наоборот, восстанавливаться и может принимать два значения:
  + CompressionMode.Compress: данные сжимаются
  + CompressionMode.Decompress: данные восстанавливаются

Если данные сжимаются, то stream указывает на поток архивируемых данных. Если данные восстанавливаются, то stream указывает на поток, куда будут передаваться восстановленные данные.

* GZipStream(Stream stream, CompressionLevel level, bool leaveMode): параметр leaveMode указывает, надо ли оставить открытым поток stream после удаления объекта GZipStream. Если значение true, то поток остается открытым
* GZipStream(Stream stream, CompressionMode mode, bool leaveMode)

Для управления сжатием/восстанавлением данных GZipStream предоставляет ряд методов. Основые из них:

* void CopyTo(Stream destination) : копирует все данные в поток destination
* Task CopyToAsync(Stream destination): асинхронная версия метода CopyTo
* void Flush(): очищает буфер, записывая все его данные в файл
* Task FlushAsync(): асинхронная версия метода Flush
* int Read(byte[] array, int offset, int count): считывает данные из файла в массив байтов и возвращает количество успешно считанных байтов. Принимает три параметра:
  + array - массив байтов, куда будут помещены считываемые из файла данные
  + offset представляет смещение в байтах в массиве array, в который считанные байты будут помещены
  + count - максимальное число байтов, предназначенных для чтения. Если в файле находится меньшее количество байтов, то все они будут считаны.

8. Для чего служит класс Path?

► предназначен для работы с именами файлов и путями в файловой системе Назначение ► выделить имя файла из полного пути ► скомбинировать для получения пути имя файла и имя каталога ► сгенерировать имя для временного файла или каталога

9. Что такое тестовый и бинарный файл. Как можно выполнить чтение и запись в эти файлы.

Текстовые – в текстовом виде, бинарные – в бинарном, любые символы кодовой таблицы.

==в.5 && в.4

+Класс **FileStream** представляет возможности по считыванию из файла и записи в файл. Он позволяет работать как с текстовыми файлами, так и с бинарными.

Сама процедура **чтения** или **записи** **бинарного** **файла** состоит из тех же шагов, что для **текстового**: **файл** связывается с файловой переменной оператором assign, затем открывается в режиме reset или rewrite, **чтение** **и** **запись** выполняются операторами read и write соответственно, по завершении работы с **файлом** его следует закрыть оператором close.

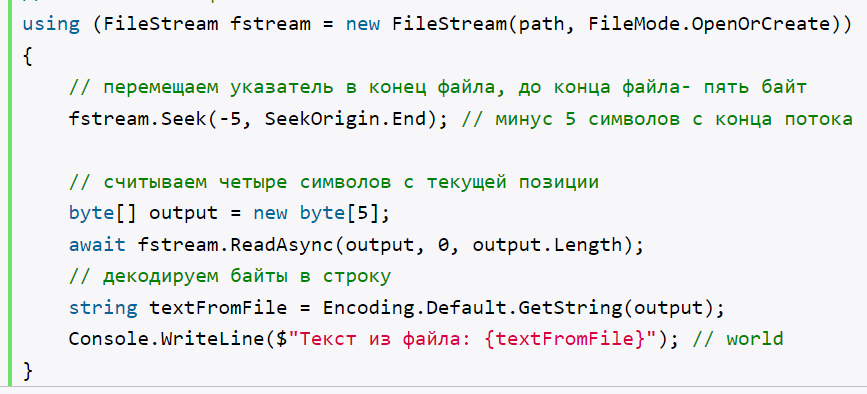
10.Что такое произвольный доступ к файлу? Приведите пример.

Нередко бинарные файлы представляют определенную структуру. И, зная эту структуру, мы можем взять из файла нужную порцию информации или наоброт записать в определенном месте файла определенный набор байтов. Например, в wav-файлах непосредственно звуковые данные начинаются с 44 байта, а до 44 байта идут различные метаданные - количество каналов аудио, частота дискретизации и т.д.

С помощью метода **Seek()** мы можем управлять положением курсора потока, начиная с которого производится считывание или запись в файл. Этот метод принимает два параметра: offset (смещение) и позиция в файле. Позиция в файле описывается тремя значениями:

* **SeekOrigin.Begin**: начало файла
* **SeekOrigin.End**: конец файла
* **SeekOrigin.Current**: текущая позиция в файле

Курсор потока, с которого начинается чтение или запись, смещается вперед на значение offset относительно позиции, указанной в качестве второго параметра. Смещение может быть отрицательным, тогда курсор сдвигается назад, если положительное - то вперед.



11.Как применяется конструкция using (не директива) при работе с файловыми потоками? Для чего ее используют.

Оператор C# - using определяет границу объекта, за пределами которой объект автоматически уничтожается. Оператор using в C# завершается, когда заканчивается блок операторов using или когда выполнение выходит из блока операторов using косвенно, например, возникает исключение.

Оператор using позволяет указать несколько ресурсов в одном операторе.

По большому счёту, работает аналогично операции try-catch.

using (получение-ресурса)  
 вложенный оператор

получение-ресурса:  
\* объявление. и инициализацию локальной переменой, ее тип реализует IDisposable, только для чтения выражение  
\* выражение, значение которого имеет тип, реализующий IDisposable